

Potential Show-Stoppers for Transactional Synchronization

Panel session, PPOPP'07, March 2007

Michael L. Scott

U Rochester

Ali-Reza Adl-Tabatabai

Intel Corp

David Dice

Sun Microsystems

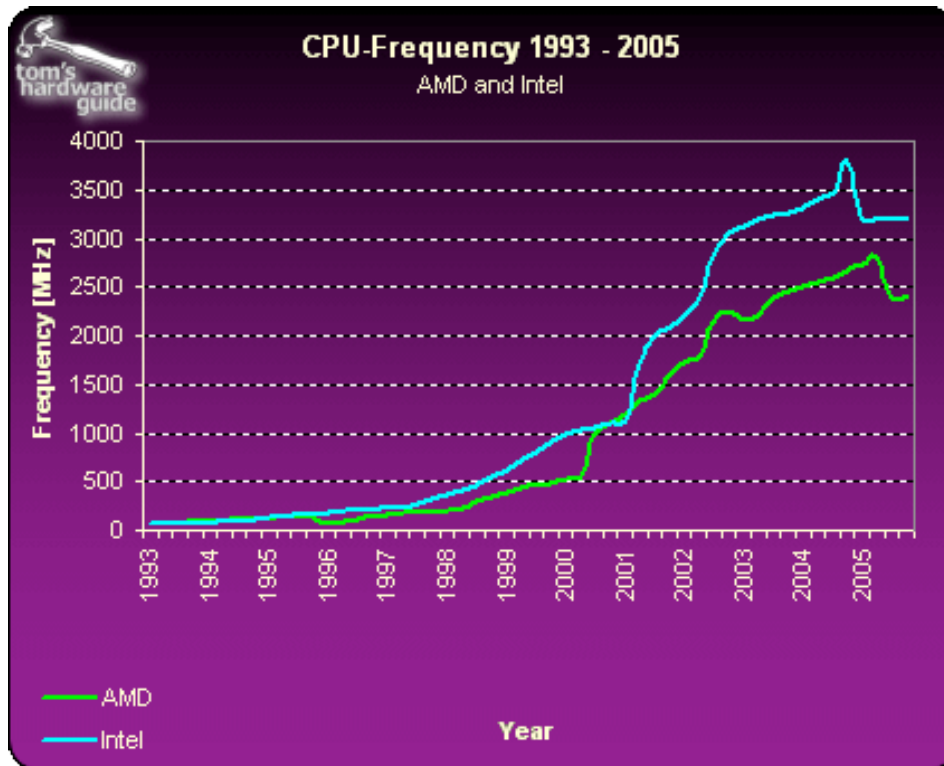
Christos Kozyrakis

Stanford U

Christoph von Praun

IBM Research

Uniprocessor Limits



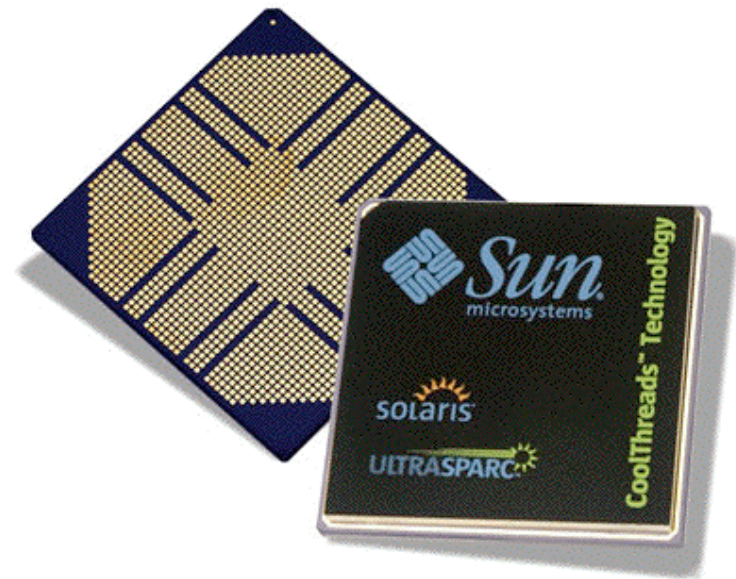
- Heat wall
- Limited ILP



<http://www.tomshardware.com/2005/11/21>

Multicore is here to stay

- Dual-processor laptops now
- Quad-core desktops
- 8-core servers
- Lots more to come
- Vendors waiting for apps



The Coming Crisis

- Parallelism common in high-end scientific computing
 - » done by experts, at great expense
- Also common in Internet servers
 - » “embarrassingly parallel”
- Has to migrate into the mainstream
 - » programmers not up to the task



<http://tfp.killbots.com/?p=wall/@wall&name=&pag=3>

What TM is

- A way to simplify some forms of synchronization — an alternative to mutual exclusion locks
- A way to improve scalability with respect to *coarse-grain* locks

What TM is *not*

- A way to make parallel programming easy
- A general-purpose synchronization mechanism
- A way to get free concurrency (or even scalability)

The basic idea is simple

- Programmer identifies atomic sections
- System serializes them, runs in parallel if it can

Some details are *not* simple

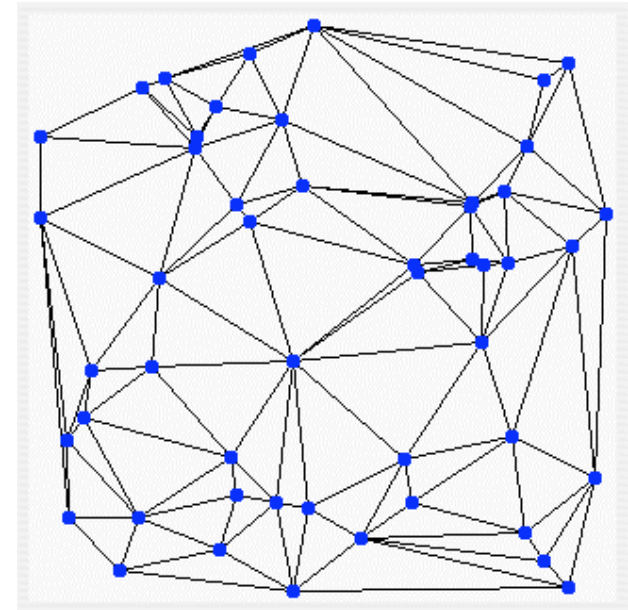
- I/O and other irreversible operations
- Open nesting: causality loops, compensating actions, high-level concurrency control
- Weak isolation, privatization
- Early release
- Condition synchronization (retry, ...)
- Alternative paths (or else, ...)
- Customizable backoff or retry policies
- Synchronizers or other cross-transaction communication
- Priorities
- Segregation of transactional and nontransactional objects or types, for the benefit of SW implementations

Not to mention

- Parallelization / identification of speculative tasks
- Ordering among transactions
- Performance tuning
 - » tools to find conflicts
 - » incentive to subdivide to avoid them
- When does this get uglier than locks?
(answer: very quickly)
 - danger of overselling

Some personal experience

- Delaunay mesh application
 - » 2500 lines of C++
 - » barrier-separated private and transactional phases
- RSTM library-based STM
 - » transactional types inherit from transactional base class
 - » access through smart pointers
- Turned out to be a lot harder than I expected



A compiler would have helped

- Hide accessors, validators
 - Generate transactional and non-transactional versions of code as needed
 - Let `this` be a smart pointer
 - Leave immutable fields in place, for safe private access; update read-only pointers as needed; support safe break/return
 - Catch loop-carried private value, potentially stale private pointer
 - Elide redundant checks
- ★ All of this is straightforward

The Bottom Line

- Keep it simple!
- Don't expect too much
- Plan on language integration and compiler support
- Do not oversell !



TRANSACT'07

**The Second ACM SIGPLAN
Workshop on Transactional Computing**

To be held in conjunction with **PODC 2007**

Portland, Oregon, August 16, 2007

Submission deadline: April 15, 2007

www.cs.rochester.edu/meetings/TRANSACT07/